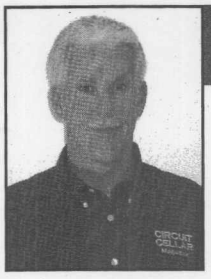

USB Easy Riders

Tom takes a closer look at USB technology and the process of upgrading applications with USB connectivity. Chips like the CP2102 make the procedure a whole lot easier.

It was way back in September 1996 when I first wrote about the then nascent Universal Serial Bus ("Oh Say Can USB?" *Circuit Cellar*, 74). Now, after 10 years (and more than 1 billion USB devices, according to www.usb.org), it's clear that USB is a boon for PC connectivity and is here to stay.

Reports of the demise of the classic serial and parallel ports are slightly, but only slightly, exaggerated. Printers, keyboards, mice, and all manner of PC-oriented gadgets are on board with USB. Now it's time for the embedded crowd to follow suit.

Porting embedded applications to USB isn't a new concept. There have been more than a few articles on the subject right here in *Circuit Cellar*. (Refer to the resources section of this article for a list of them.) Nevertheless, time and silicon march on, so let's take a look this month at the latest and greatest.

GOING, GOING, GONE?

But let's start with some not so great. What's up with USB On-the-Go? Not much, as far as I can tell.

To refresh your memory, the USB On-the-Go (OTG) initiative was announced to great fanfare way back in 2001. The idea behind OTG has been to evolve USB beyond its PC-centric host/device architecture to a more democratic peer-to-peer device/device nirvana. OTG proponents look forward to the day when your cell phone, PDA, MP3 player, etc. can talk directly to each other without needing a PC in the loop.

Sounds good, but don't hold your breath. If you poke around the 'Net, you'll find all manner of press releases in the years since the announcement. The annual PR ritual invariably includes a prediction that USB OTG-enabled devices will be "on the shelf" by "the end of this year" (i.e., whatever year it was when the hopeful hype was penned). There may be some shelves somewhere with OTG on them, but they aren't the ones at my local electronics emporium.

Sure, OTG makes sense from 50,000'. From that lofty vantage point, oxygen deprivation-induced euphoria tends to make a lot of ideas look good, including the 99% that go nowhere. Only from a much lower altitude do the messy details, and the devils that come with them, become apparent.

Yes, the first OTG-capable chips are peeking out from behind the curtain. Usually, the availability of silicon says a standard's time has arrived. But in OTG's case, I think it just means there's no more avoiding the troubling issues that remain. Chief among these is the fundamental assumption that universal device-to-device connectivity is a good thing. As the old saying goes, "where you stand is where you sit." For instance, consider one example of a problem OTG is supposed to fix. Many PDA users buy full-size keyboards for entering large amounts of data. And each brand of PDA typically has its own proprietary keyboard offering. Wouldn't it be nice if every PDA user could instead choose from a variety of standard keyboards without being tied to one brand? The answer is yes if you're Joe-PDA keyboard user. But what if you're the PDA manufacturer? Do you really want your valuable add-on keyboard monopoly destroyed by cheapo clone keyboards?

In fact, the OTG crew recognized the issue with provision for the so-called target peripheral list (TPL). In essence, this concedes the fact that the aforementioned PDA manufacturer may prefer to support only their own add-on keyboards.

Another oft-cited example of the need for OTG is the idea of a digital camera being able to connect directly to a printer instead of having to two-step a picture file from the camera to the PC, and then from the PC to the printer. But instead, how about just giving the printer itself limited USB host capability? Check out the photo printers down at the emporium, and you'll see that's exactly what's happening (look for the PictBridge logo).

Remember that the original success of USB was only by virtue of a huge up-front investment by Intel. They put USB connectors on the motherboard long before there was much of anything to plug in. That made sense for Intel, because it made PCs easier to use (more sales, more add-ons, less support). Oh, yeah, because they don't make the gadgets that plug into USB, the commoditization of add-ons was no skin off their nose.

So, who's going to volunteer to drive USB OTG? Given its goal to eliminate the PC as a middleman, I doubt Intel will be leading the charge. What other valiant supplier will take the risk of being first, and maybe last, on the block? And of that short list, who has the muscle to move the market?

Think I'm being overly pessimistic about OTG? Well, check out the market predictions in Eric Huang's "USB On-The-Go Overview," which was presented at an August 2004 USB OTG training seminar (see Figure 1).[1]

The OTG projections look rather yawn-inspiring if you ask me.

KEEP IT SIMPLE

Rather than spending time on tomorrow's problems of tomorrow, I'm more interested in ways to get a simple, not to mention home-brew, gadget hooked up to USB today, preferably with a minimum of muss and fuss. One option is to use one of the many USB-enabled MCUs as the basis for your design. The good news is that there is a large and ever-growing list of parts to choose from. Nevertheless, your choice of platform is limited (i.e., the number of USB-enabled MCUs is a tiny fraction of the entire MCU universe).

Another concern is that you may have to learn more than you want to about the innards of USB, even presuming a lot of support from the chip supplier (sample drivers, application notes, etc.). At the least you'll have to delve into the subject enough to assure yourself that handling the USB stuff doesn't compromise your application's performance and robustness.

After you decide to sell a USB gadget to the unwashed masses, the PC-side driver becomes a huge stumbling block. Writing the driver is one thing, but it's a problem that pales in comparison to the fact that you're now in the PC software business whether or not you like it. Calls in the middle of the night from brand X PC users. Upgrades and bug fixes. Microsoft sneezes and you catch pneumonia. Yechh!

One good idea is to piggyback on the drivers Microsoft already provides. The simplest and most generic is to impersonate a keyboard or mouse as a member of the human interface device (HID) class. It might be tempting to go for a higher performance class (such as mass storage and, most recently, video), but they come with a lot more baggage, overhead, and complexity.

I've concluded that, for now, one of the best ways (certainly the easiest) to get on board is a simple USB-to-serial adapter solution. In conjunction with a virtual COM port (VCP) driver on the PC, a converter can allow serial port-based applications to hook up to USB, yet run as they are with no hardware or software changes. If you're connecting an existing design that includes an RS-232 transceiver and the familiar DE-9 or DB-25 connector, there are plenty of off-the-shelf converter dongles to choose from.

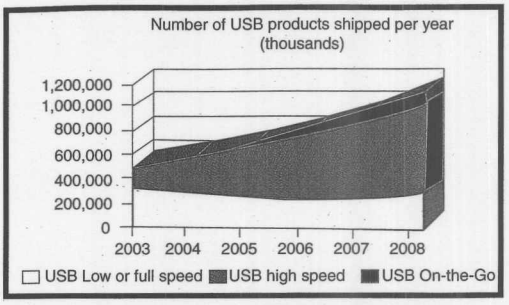

Figure 1—*Ready, set, don't go. USB On-the-Go faces a long and winding road before it achieves anything near the popularity of the original. (Source: In-Stat/MDR February 2004)*

Besides compatibility and simplicity, using a converter offloads all of the USB processing from your host MCU. And now you have the luxury of choosing any MCU with a UART (hardware or bit-banged) for your application processing. If you have the luxury of making a PCB, you can streamline things a lot by connecting a USB-to-serial adapter directly to your UART, eliminating the need for two RS-232 transceivers and connectors, and presenting a clean USB-only facade to the outside world.

Kudos to Future Technology Devices International (FTDI) for moving the USB-to-serial adapter forward. Besides highly integrated chips, which you'll find under the hood of many stand-alone converters, they deliver a royalty-free VCP driver. They also support a direct driver option that comes with the requisite dynamic link library (DLL) and application program interface (API). In short, FTDI bites the PC software bullet (i.e., maintenance, support, bug fixes, etc.) so you don't have to.

FTDI's latest chip, the FT2232C, incorporates a lot of features (see Figure 2). Like earlier versions (e.g., FT232BM), it includes practically everything you need for a USB serial solution. On the USB side of the chip, you just need to provide a 6-MHz clock (boosted to the 48-MHz clock USB required by an integrated PLL) and a handful of discretes and power logic depending on whether your USB gadget is bus-powered or self-powered. Bus-powered designs are served with an on-chip voltage regulator that converts the nominal 5-V USB bus voltage to 3.3 V.

You can choose to add a small serial EEPROM chip if you want to override the default USB settings (vendor ID, product description, transfer mode, etc.). The EEPROM connects directly to the '2232C and FTDI provides a utility that allows programming it via USB. The EEPROM can be shared with a local MCU by virtue of the fact that the '2232C EEPROM interface pins (i.e., chip select, clock, and date) are tristated when the chip is held in reset.

It's on the MCU inter-

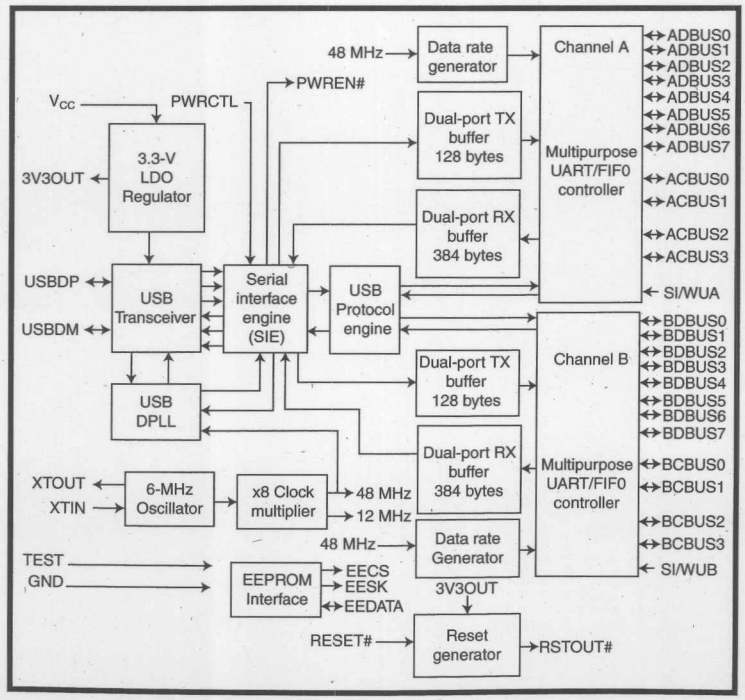

Figure 2—*FTDI started the USB-to-serial adapter chip craze. Now they're upping the ante with the two-channel FT2232C.*

face side of the part that the '2232C breaks new ground, starting with two completely independent ports. The power-up default configuration has both ports configured as PC COM port-style UARTs including a complete set of modem control lines (RTS, CTS, etc.). The big news is that the ports also can be configured in a variety of other modes, including FIFO, MCU bus emulation, and clocked serial. These additional modes enable higher performance across a broad range of applications. In some cases they may even eliminate the need for an MCU altogether.

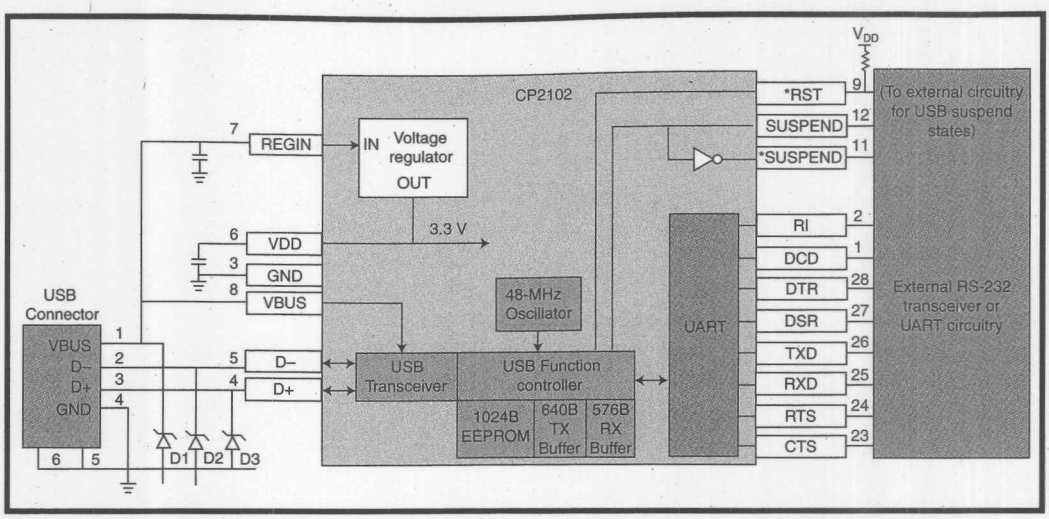

Figure 3—*Silicon Laboratories's CP2102 represents a new low in USB-to-serial adapter chips: lower chip count, lower power, and less board space.*

For instance, to support the migration of PC-based emulators, downloaders, and flash memory programmers to USB, the '2232C Clock Serial mode provides the basis for USB-JTAG emulators. Another option described in the datasheet is an optoisolated connection to serve the growing stable of industrial and scientific add-ons (e.g., data logging and digital oscilloscope).

KEEP IT SIMPLER

Witnessing the success of FTDI, it's no surprise to see competitors emerge. As an aside in last month's column, "'51 Favorites," I noted that Silicon Labs's (formerly Cygnal) '51 evaluation board exploited their serial-to-USB adapter chip, the CP2102 (see Figure 3). Let's take a closer look.

In terms of basic functionality, the CP2102 is similar to FTDI's FT232BM.

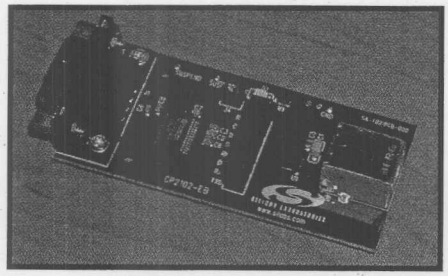

Photo 1—*The Silicon Laboratories CP2102 evaluation board bridges the gap between the old (RS-232) and the new (USB). Notice the relative component count and floor space required for each, which only reinforces the conclusion that it's time to make a change.*

Both are minimalist converters featuring the aforementioned USB direct connection on one side and a COM port-oriented UART with a full complement of modem controls on the other. Like the FTDI parts, royalty-free drivers (both Virtual COM Port and Direct DLL) are available. However, although similar in terms of basic functionality, the CP2102 delivers some key cost-saving advantages when it comes to design-in.

In what I believe may be a first, the CP2102 integrates the 48-MHz USB clock on-chip, completely eliminating the need for an external clock source (crystal, resonator, etc.). This is quite an accomplishment, because the on-chip clock must be accurate enough to meet USB specs, a nontrivial challenge considering aging and varying environmental factors (e.g., temperature and voltage).

The CP2102 also integrates the EEPROM required for nondefault USB settings that would otherwise require an external chip (as is the case for the FTDI parts). Like FTDI, Silicon Labs provides a PC-based utility for programming the EEPROM via USB.

Although both the Silicon Labs and FTDI parts have a similar pin count (28 pins for the CP2102 and 32 pins for the FT232BM), the CP2102 at only 5 mm × 5 mm has a much smaller footprint than the 9 mm × 9 mm FT232BM. Another key spec difference is the –40° to 85°C temperature range for the CP2102 versus 0° to 70°C range for the FT232BM.

Price projections always come with caveats related to volume, timing, channel (direct versus distribution), and your negotiating skill, not to mention the challenge posed by editorial lead times. That being said, as I write this column, it appears pricing for the two parts is similar at about $5 in low quantity dropping to $2.50 in high volume. Obviously, given its integrated clock and EEPROM, this gives a system cost advantage to the CP2102.

GETTING VIRTUAL

I decided to give the CP2102 evaluation board a look-see. The $49 board implements the usual USB-to-RS-232 adapter function in two chips (the CP2102 and a Sipex RS-232 transceiver), the requisite connectors, and little else (see Photo 1).

The CD that comes with the kit includes Virtual COM port drivers for Windows (98 through XP), Mac (OS-9 and OS-X), and Linux 2.40. The driver installation seemed a bit quirky. After plugging in the board to the PC USB port, I went through two sessions with the Windows New Hardware Wizard. First it installed the CP210X USB composite device and then the CP210x USB-to-UART bridge controller. All went well, and I used the Windows device manager to verify that the CP2102 board was assigned to an open COM port slot.

Thanks to the COM port impersonation, initial checkout is easy using a simple terminal emulator such as HyperTerminal, which comes with Windows. However, I've always found that program, admittedly designed with

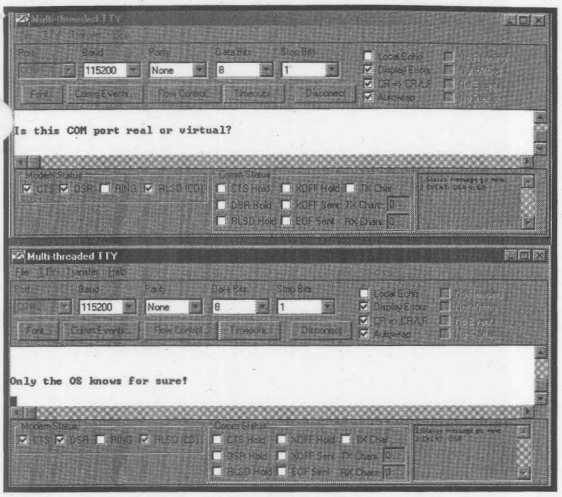

Photo 2—*After installing the virtual COM port (VCP) drivers, you can perform initial CP2102 loop-back tests on a single PC by running one copy of a terminal emulator talking to the virtual COM port and one copy talking to the real COM port.*

modems in mind, a little clunky when it comes to direct RS-232 communications. Instead, I usually use an emulator called MTTTY, which is a sample program that comes with a Microsoft serial port application note, itself useful reading.[2]

To start, I set up a simple loop-back configuration, connecting the PC's real COM port to the CP2102 board's virtual COM port. I had to admit some concern that between Windows, MTTTY, and the Silicon Labs VCP driver, the PC might get all choked up trying to talk to itself. But, as shown in Photo 2, I was able to connect each copy of MTTTY to its COM port (real and virtual, respectively) and establish communication.

Next, I moved up to true computer-to-computer communication. The configuration consisted of a new and fast XP PC and an old slow Windows 98 laptop. I experimented in both directions with each computer taking a turn connecting by real (RS-232) and virtual (USB) COM ports. This confirmed that both versions of the VCP driver (i.e., XP and Windows 98) were functional and could talk with each other.

Next, I exercised the various handshaking features, both hardware (RTS/CTS and DTR/DSR) and software (XON/XOFF). As expected, with handshaking disabled, I was able to induce overrun errors when transferring a big file from the fast computer to the slow laptop (see Photo 3). With handshaking enabled the overrun problem went away, and the file transferred successfully. This serves as a reminder that even though the CP2102 does an admirable job, it can't work miracles such as making a slow computer fast.

Data rates are another possible gotcha. The CP2102 datasheet touts the chip's ability to deliver data at up to 1 Mbps. Besides the fact such a high rate would no doubt push the envelope of what a particular PC can handle, most existing applications don't know about anything above and beyond the standard data rates (e.g., 300 bps to 115.2 kbps).

To add a twist, the workaround is different depending on your specific PC operating system. For Windows 98, it's possible to access the UART control registers directly to program the desired data rate. However, XP doesn't allow application software to have direct access to chips. (Stop me before I crash again.) The recommended hack is to change the clock divisor factors stored in the USB .INF file to allow a non-standard data rate to impersonate a standard one. For example, you could set the entry corresponding to 115.2 kbps with the clock divisor factors corresponding to 1 Mbps. Then, when an application specifies 115.2 kbps, the rate is actually set to 1 Mbps behind the scenes.

Although it's tempting to succumb to the illusion of real COM ports, don't forget there's a lot that goes on between the two ends of the virtual connection. Consider an example application that takes direct control of the modem control lines, perhaps to monitor a switch closure or drive an LED. With a real COM port, it's possible to get a reasonably accurate handle on the timing. But a virtual COM port introduces a degree of uncertainty due to the fact the actual I/O timing depends on USB scheduling beyond the control of your application.

I wrote a simple test program in BBC BASIC to demonstrate the difference (see Photo 4). Refer to the "London Calling" sidebar (page 84) for more information about BBC BASIC. All the program does is sit in a tight loop toggling the RTS line as fast as possible. Sure enough, probing with an oscilloscope revealed the real RTS line was switching about 100 times faster than the virtual one (e.g., 20 kHz versus 200 Hz).

The oscilloscope session also served as a cautionary tale that has nothing to do with the CP2102 directly, but rather the RS-232 interface. I noticed the voltage swing was twice as high on the RTS output from the PC (e.g., ± 10 V) compared to the output from the USB

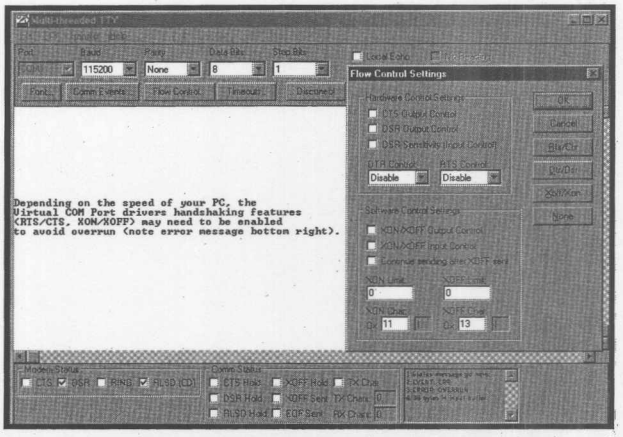

Photo 3—*The CP2102 includes a healthy amount of buffer RAM to support high data rates, but take care not to overdo it. In this case, transferring a large file from a fast to a slow PC caused overruns on the latter. Use hardware and software handshaking (or lower the data rate) to make the problem go away.*

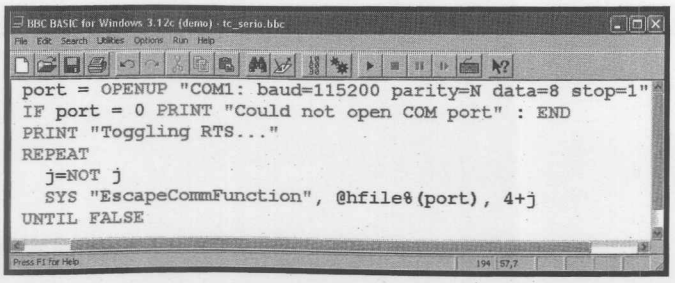

Photo 4—*It may not matter in most applications, but this simple program to toggle the RTS line reveals there are low-level timing differences between a real and virtual COM port.*

bus-powered RS-232 transceiver on the CP2102 board (e.g., ±5 V). I didn't have any problems myself, but it isn't hard to imagine scenarios (cabling, noise, using an RS-232 line as a power source) in which the difference could lead to glitches and head scratching.

JUMPSTART

Chips like the CP2102 minimize the pain of upgrading your application with USB connectivity. Yes, you could just stick with RS-232 and rely on users to buy their own USB-to-RS-232 converter dongle. That may seem like an easy way out, but does it really make sense?

Consider the fact that by the time you add an RS-232 transceiver and DE-9 connector, there will be little (if any) cost, size, or power consumption advantage compared to a USB-to-serial adapter chip and connector (the latter is optional as with a USB keyboard or mouse).

Presenting a USB-only face to the world also gives your design the credibility associated with products from major players who have the wherewithal to roll their own USB solution. It also delivers cross-platform plug ability (e.g., PC, Mac, Linux, etc.) that a serial or parallel solution can't.

Finally, the hope that leaving the USB issue to the user will relieve you of support headaches is likely an illusion. The fact is that if the user has a problem using your RS-232 gadget with an external USB converter dongle, you have a problem too. Chances are you'll still get a call if a problem arises, and you may end up having even more USB-related support headaches (e.g., multiple vendors dongles, drivers, etc.) than if you were to bite the bullet and take charge of the USB side of the equation yourself.

There's no doubt that the time has come to ditch the past and get on the USB bandwagon. The only question I have is, What the heck should I do with my closet full of soon-to-be-obsolete nine-pin (and even older 25-pin) RS-232 stuff? Oh well, that's progress.

Tom Cantrell has been working on chip, board, and systems design and marketing for several years. You may reach him by e-mail at tom.cantrell@circuitcellar.com.

London Calling

I wanted to write some simple programs to check out the CP2102. Given the USB connection, that meant programming the PC instead of a simple SBC that I would typically use. However, I hadn't been keeping up with the times, especially when it came to today's complicated PC programming mega-suites. Studying up on the subject a bit, I quickly convinced myself that the fancy tools out there were not only overkill, but also way too bloaty to get my hands around in short order. Oh, for the good old days when PCs came with a simple BASIC built in.

Almost as a matter of whimsy, I punched "BASIC for Windows" in Google, which faithfully reported 25 million hits. Sigh. But wait, right there on the first page, I saw three links for a "BBC BASIC for Windows." What the heck? No harm in clicking.

BBC BASIC goes back to the roots of personal computing in the United Kingdom, even before the IBM PC days, a story filled with names from the past like Acorn (progenitor of ARM), Sinclair, and even CP/M. I enjoyed the trip down memory lane, but even better the fact I came away with a free download evaluation version of BBC BASIC for Windows. Head on over to www.cix.co.uk/~rrussell/products/bbcwin/bbcwin.html and check it out.

Compared to the original BASICs of yore, BBC for BASIC is extremely improved. You can have long variable names, and line numbers are only required for statements that are the target of branches. Fortunately, you don't need many of those because it has a good complement of structured programming features (e.g., procedures with local variables), making for readable code instead of GOTO spaghetti.

Otherwise, BBC BASIC is an interesting mix of powerful features (32-bit integer and 32- or 64-bit floating-point math) and eclectic ones (obscure graphics commands from the days of Teletext and EGA). The GUI is simple and clean, just the thing for dashing out some one-liners, and it has excellent built-in HELP. Best of all, it came with a healthy complement of serial I/O features that were easy to use and worked properly on both my XP and Windows 98 PCs.

The bad news is the free evaluation version is limited to 8 KB of program space. The good news is that because it's a tokenizing interpreter, 8 KB goes further than you might imagine. For example, the program I wrote to toggle the RTS line only consumed a couple of hundred bytes. Or go ahead and splurge on the full version, which is only £29.99 and includes a compiler (generate an .EXE file). Jolly good show, I say!

REFERENCES

[1] E. Huang, "USB On-The-Go Overview," www.usb.org/developers/onthego/san_jose_otg_presentations.zip.

[2] A. Denver, "Serial Communications in Win32," 1995. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnfiles/html/msdn_serial.asp.

RESOURCES

J. Axelson, "USB Chip Choices: Finding a Peripheral Controller," *Circuit Cellar*, 120, July 2000.

J. Bachiochi, "USB DMX," *Circuit Cellar*, 172, November 2004.

———"USB in Embedded Design," *Circuit Cellar*, 165, April 2004.

F. Eady, "Mission Possible: Achieve Cheap USB Connectivity," *Circuit Cellar*, 157, August 2003.

J. Pollard, "USB, FTDI Style," *Circuit Cellar*, 132, July 2001

USB Implementers Forum, www.usb.org.

SOURCES

FT232BM and FT2232C Chip
Future Technology Devices International
www.ftdichip.com

CP2102 Chip
Silicon Laboratories
www.silabs.com